

# STK-Mesh Example Computation Setup and Parallel Execution

## Carter Edwards, **Todd Coffey**, Dan Sunderland, Alan Williams



Sandia National Laboratories



# Outline

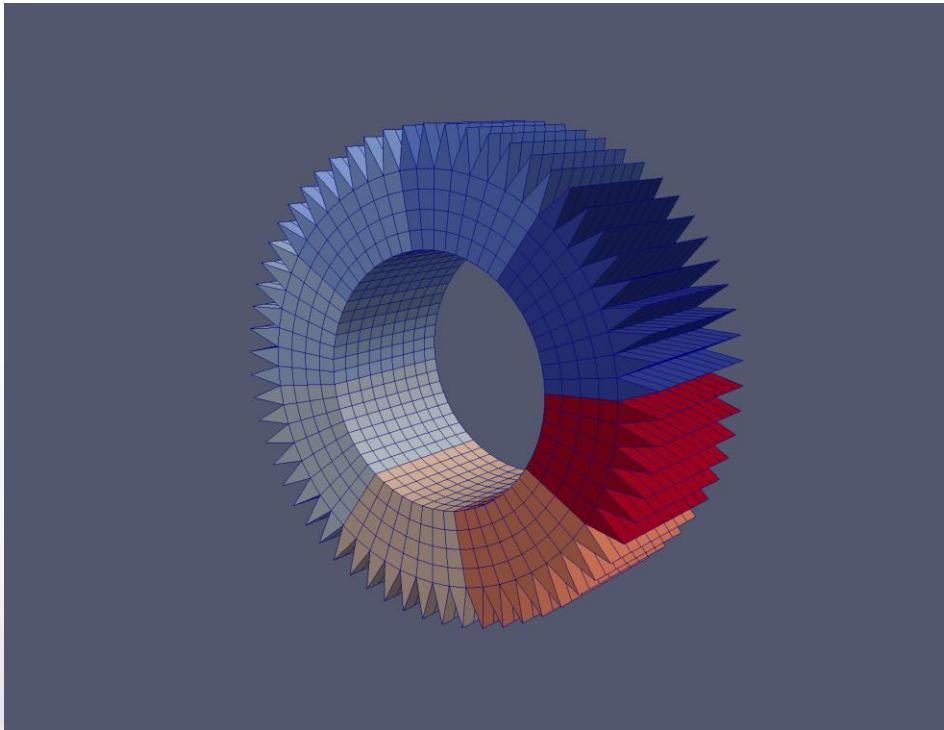
---

- Overview of Gear Fixture & features
- Setup of Meta Data
- Setup of Bulk Data
- Computational Kernel
- Movie



# Description of Gears Demo

- Basis for movie at Super-Computing 2010
- Example is in STK sources and is used for testing



# Overview of Gear Fixture

- Located in source:
  - <http://trilinos.sandia.gov/>
    - packages/stk/stk\_mesh/stk\_mesh/fixtures/GearsFixture
    - packages/stk/stk\_mesh/stk\_mesh/fixtures/Gear
    - packages/stk/stk\_performance\_tests/stk\_mesh/GearsSkinning
- Cylindrical gear body made up of Hexahedron<8> elements
- Gear Teeth made up of Wedge<6> elements
- Parallel distribution of elements and nodes
- Rotation of gear



# Gear Meta Data

- **Declare the following parts of the mesh**
  - Cylindrical coordinates part
  - Hex part
  - Wedge part
- **Declare the following fields:**
  - Cartesian coordinates field ( $x,y,z$ ) on nodes
  - Displacement field ( $\Delta x, \Delta y, \Delta z$ ) on nodes
  - Translation field on nodes
  - Cylindrical coordinates field ( $r,\theta,z$ ) on nodes
- **Restrict the fields to parts**



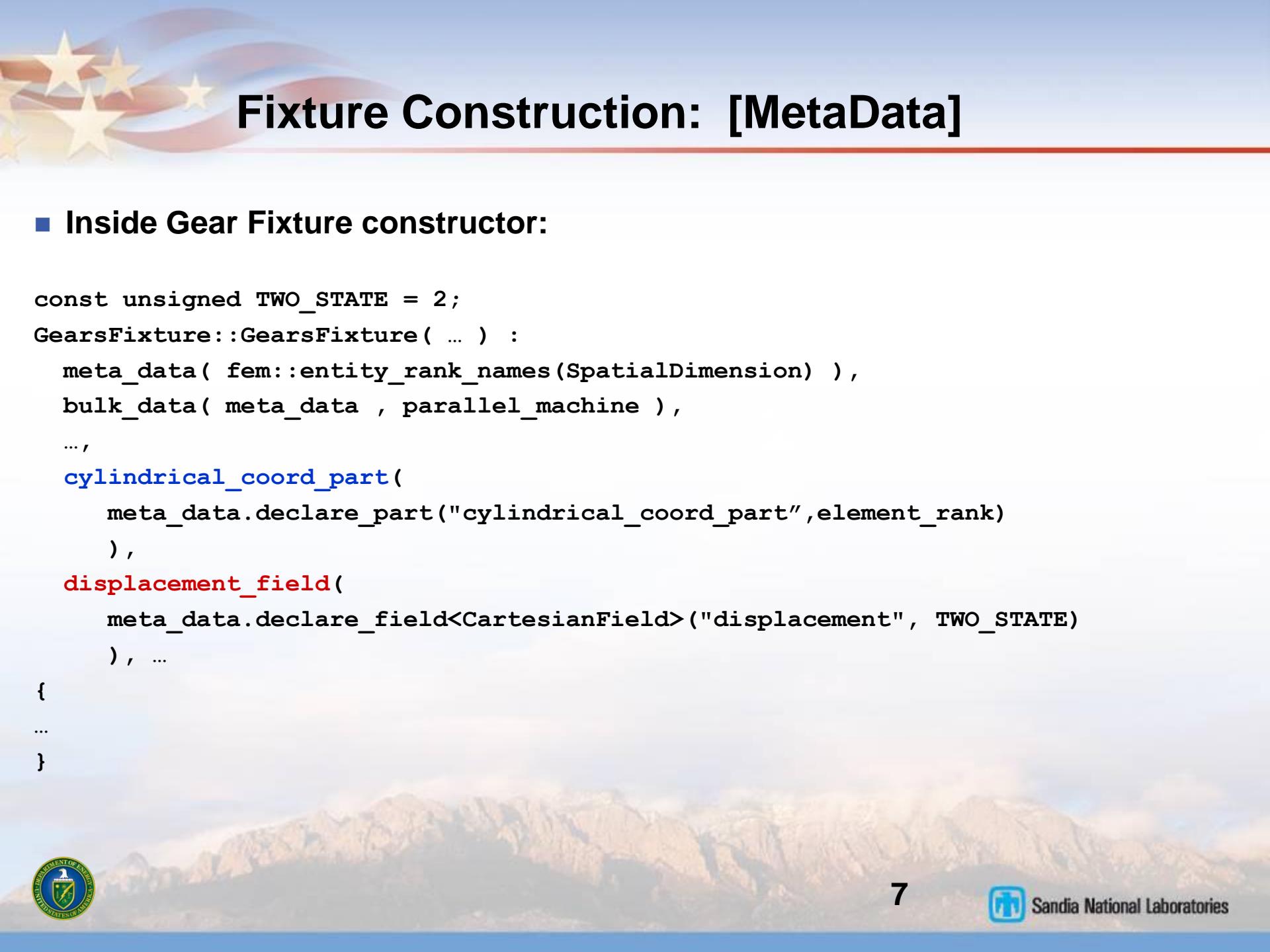
# Fixture Declaration of Meta Data

```
typedef Field< double , Cylindrical>    CylindricalField;
typedef Field< double , Cartesian>      CartesianField;

class GearsFixture {
public:
    MetaData    meta_data;
    BulkData    bulk_data;
    ...
    Part & cylindrical_coord_part;
    Part & hex_part;
    Part & wedge_part;
    CartesianField   & cartesian_coord_field;
    CartesianField   & displacement_field;
    CartesianField   & translation_field;
    CylindricalField & cylindrical_coord_field;
    ...
};

};
```





# Fixture Construction: [MetaData]

- Inside Gear Fixture constructor:

```
const unsigned TWO_STATE = 2;  
GearsFixture::GearsFixture( ... ) :  
    meta_data( fem::entity_rank_names(SpatialDimension) ),  
    bulk_data( meta_data , parallel_machine ),  
    ...,  
    cylindrical_coord_part(  
        meta_data.declare_part("cylindrical_coord_part",element_rank)  
    ),  
    displacement_field(  
        meta_data.declare_field<CartesianField>("displacement", TWO_STATE)  
    ), ...  
{  
...  
}
```





# Restrict Fields to Parts: [MetaData]

- Field Restrictions: Entities in the part will have the field
- Example for cylindrical coordinate field and part:

```
GearsFixture::GearsFixture(...) : [prev slide] {  
    put_field(  
        cylindrical_coord_field,  
        node_rank,  
        cylindrical_coord_part,  
        SpatialDimension  
    );  
    ...  
}
```





# Induced Part Membership

## Simplified Mesh Maintenance

- How to apply a field to nodes that are connected to elements of a particular type?
  - Option A: Manually
  - Option B: Automagically
    1. Put elements in a Part of that type
    2. Restrict the nodal field to that part
    3. All nodes in closure of elements in this part will get this field
- Cylindrical Part limited to Elements
- Cylindrical coordinates Field limited to Nodes
- Cylindrical coordinates field restricted to cylindrical part





# Gear Bulk Data Operations

- **Create Hex entities for gear body**
  - Place Hex entities in Hex part and Cylindrical part
  - Declare relations to nodes
- **Create Wedge entities for gear teeth**
  - Place Wedge entities in Wedge part and Cylindrical part
  - Declare relations to nodes
- **Initialize field data**
- **Skin mesh with faces**
  - Skinning function in fem



# Initialize Field Data (entity-wise)

- Loop over nodes
- Assign Cartesian coordinates from cylindrical coordinates

```
loop over nodes {  
    const double rad = ... ;  
    const double angle = ... ;  
    const double height = ... ;  
  
    double * const cartesian_data = field_data( cartesian_coord_field , node );  
  
    cartesian_data[0] = rad * std::cos(angle);  
    cartesian_data[1] = rad * std::sin(angle);  
    cartesian_data[2] = height;  
}
```





# Gear Rotation Computation (bucket-wise)

- Recall: Select buckets, iterate over buckets, compute

```
Selector select = cylindrical_coord_part &
                  ( locally_owned_part | globally_shared_part );

BucketVector selected_node_buckets;

get_buckets(
    select,
    bulk_data.buckets(Node),
    selected_node_buckets
);
```



# Gear Rotation Computation (bucket-wise)

```
for (BucketVector::iterator b_itr = selected_node_buckets.begin();  
     b_itr != selected_node_buckets.end();  
     ++b_itr) {  
    Bucket & b = **b_itr;  
  
    const BucketArray<CartesianField>  
        old_coordinate_data( cartesian_coord_field, b );  
    BucketArray<CartesianField>  
        displacement_data( displacement_field.field_of_state(StateNew), b );  
    ...  
  
    for (size_t node_index = 0; node_index < b.size(); ++index) {  
        // Compute new and old coordinate data and assign to displacements:  
        displacement_data(0,index) = new_coordinate_data[0] - old_coordinate_data(0,index);  
        displacement_data(1,index) = new_coordinate_data[1] - old_coordinate_data(1,index);  
        displacement_data(2,index) = new_coordinate_data[2] - old_coordinate_data(2,index);  
    ...  
    }  
}
```



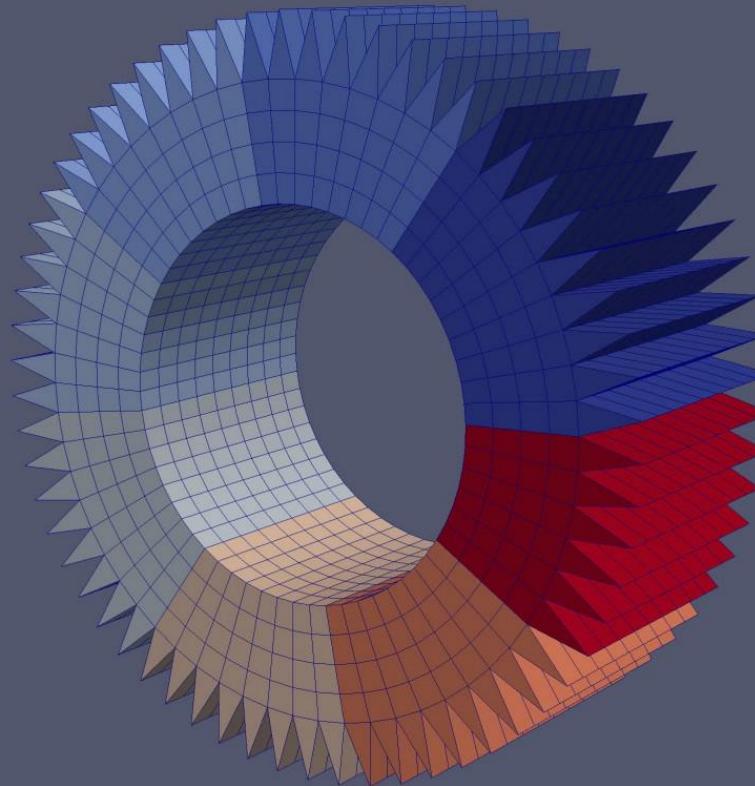


# Hybrid Parallelism

- Alan's talk will cover hybrid parallelization of this example
- Basic Idea: Parallel For applied to bucket loops
- Also see the following example:  
`stk_usecases/app/UseCase_blas_algs.hpp`  
`stk_usecases/app/UseCase_blas.cpp`



# Gears Demo (stk\_mesh, epu, conjoin, paraview)





# Supplemental Material





# MetaData Parts

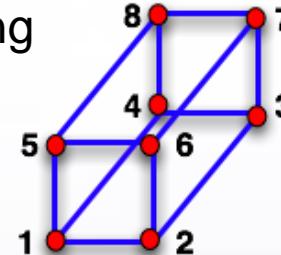
- Basic container for Entities
- Parts have names for I/O and diagnostics
- Superset/Subset feature
- Universal Part contains all other parts and all entities
- Parts can be restricted to an Entity Rank (like Element) induced part membership
- Defines data-decomposition of Buckets
  
- Selectors apply set logic to Parts to choose Buckets
  
- Fields are defined on Parts
- Fields are applied to Entities through Part membership
- Cell Topology is defined by Part membership also, but no induced topologies



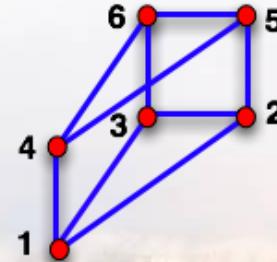
# Cell Topology

- **Cell Topology defines topology of entity**
  - Sub-topologies of sides
  - Normals
  - Prescribed number of relations downward
- **Order of downward relations is important for normals**

- Hexahedron<8> node ordering



- Wedge<6> node ordering





# MultiState Fields

## ONE\_STATE and TWO\_STATE

- States allow for “current” and “new” (and others)
- Pointer swapping eliminates copies of data
- Explicit declaration of field state
- BulkData.update\_field\_data\_states()  
rotates the field data of multi-state fields in a ring
- Two state fields can swap the fields
- Both states need to be initialized





# Induced Part Membership

- Cylindrical Part limited to Elements
- Cylindrical coordinates Field limited to Nodes
- Cylindrical coordinates field restricted to cylindrical part
  
- Rank limited parts use induced part membership
- All downward related entities are induced into this part
  
- This means all the nodes related to elements in the cylindrical part will have the cylindrical coordinates field
  
- Universal part contains all entities  
fields restricted to the universal part are applied to all entities  
(in this case, all node entities)





# Bulk Data Construction in Code (Serial)

- Generate new entities with valid global IDs

```
std::vector<size_t> requests(meta_data.entity_rank_count(), 0);
requests[Node] = num_nodes;
requests[Element] = num_elements;

bulk_data.generate_new_entities(requests, gear_entities);
```



# Hex Creation

```
{  
    std::vector<Part*> add_parts, remove_parts ;  
    add_parts.push_back( & cylindrical_coord_part );  
    add_parts.push_back( & hex_part );  
  
    for ( size_t ir = 0 ; ir < rad_num -2 ; ++ir ) {  
        for ( size_t ia = 0 ; ia < angle_num; ++ia ) {  
            for ( size_t iz = 0 ; iz < height_num -1 ; ++iz ) {  
  
                Entity & elem = get_element(iz, ir, ia);  
                bulk_data.change_entity_parts(elem, add_parts, remove_parts);  
  
                const size_t ia_1 = ( ia + 1 ) % angle_num ;  
                const size_t ir_1 = ir + 1 ;  
                const size_t iz_1 = iz + 1 ;  
  
                Entity * node[8] ;  
  
                node[0] = & get_node(iz , ir , ia_1 );  
                node[1] = & get_node(iz , ir , ia ) ;  
                node[2] = & get_node(iz_1, ir , ia ) ;  
                node[3] = & get_node(iz_1, ir , ia_1 ) ;  
                node[4] = & get_node(iz , ir_1, ia_1 ) ;  
                node[5] = & get_node(iz , ir_1, ia ) ;  
                node[6] = & get_node(iz_1, ir_1, ia ) ;  
                node[7] = & get_node(iz_1, ir_1, ia_1 ) ;  
  
                for ( size_t j = 0 ; j < 8 ; ++j ) {  
                    bulk_data.declare_relation( elem , * node[j] , j );  
                }  
            }  
        }  
    }  
}
```

